Due Date: At the START of class, Wednesday, June 19, 2002.

This assignment is simply to write the user manual for the project you started in Assignment #1. See the user manual example table of contents for guidance as to what you will need to include (it is in Section 6.11 of the lecture notes). Also, follow the Documentation Planning Guide carefully. And there are some completed student user manuals in the SFU Library Reserve Room.

Note: Before modifying any of your release pages from Assignment #1, or the requirements specification, you are REQUIRED to turn on Microsoft Word change tracking. Vertical bars will show up in the margins and indicate what portions of the document have been changed. This will allow us to not bother re-reading all the parts of the requirements spec again. Try Tools > Track Changes > check Track Changes While Editing > then select Changed Lines = left border, and de-select all the other change annotations to 'none' or 'hidden'.

In order to simplify this and later assignments (to reduce the workload in Cmpt 275), you may assume you are writing a manual for a prototype of the system which will have the following specific simplification(s):
Simplification #1) The aircraft model and flight objects can be reduced to 2NF. i.e. there will be no models. Every time a user creates a flight the user will need to enter aircraft length and max weight attributes. This saves you programming the whole model class and model file code.
Simplification #2) You needn't program delete customer. You can, if you want, program a 'special delete' which deletes a customer but does not maintain referential integrity. Or a special maintenance function that deletes all customers. But give these menu items names that use the word "special" (e.g. 'special delete'). This may be handy during testing, rather than having to stop the program and use the operating system to empty or delete the customer file. Feel free to put any 'special' commands you want in the code (and optionally in the manual); they don't look great but are really helpful for testing. In contrast, note that the cargo item delete and the flight delete must be elegant and handle things correctly. Also note that if you have a special maintenance commands in the code, we will not insist that it show in the manual, and if it does show in the manual it does not have to be well documented. Put such commands at the end of menus and submenus so that the special code can be commented out for production yet not change any of the numbering of the remaining menu items.
Simplification #3) Depending on how you design the assign operation, you may not need the 'query for unassigned cargo' operation. If the assign operation lists unassigned cargo to pick from, then you do not need the cargo item 'list all unassigned' menu operation.
Note #1) For the flight operations, you need a create, elegant delete, query space remaining, print flight load report, and 'reassign all cargo to different flight' operation. (In assignment #1 the latter was assign new model to flight, but since we no longer have models, this operation should be changed to re-assign cargo to different flight.) I was going to allow you to reject the reassign operation if all the cargo will not fit on the other flight, but it is so easy to program unassigning any excess cargo that doesn't fit that I am now insisting that this reassign operation be elegant. Don't forget that you need the regular assign operation. I am not sure whether you consider this an cargo item operation or a flight operation; you will have to design the menu tree and the look and feel for this operation.
Note #2) Since you are going to have to write functions that move (i.e. deassign and assign) an item to a new flight, it should be very simple to provide a user with a unassign operation that simply removes one item from a flight into the unassigned state, and therefore makes more room available on the flight.

You do NOT have to update the Requirements Spec to correct for spelling and grammar, etc. But you should update it to fix the keys, attributes, and cardinalities of the objects, to fix any problems in your DFD, and to fix the state diagram for the reservation, as per the marking by the instructor or TA. And fix any format and history/table problems. We may not get the marked Requirements Spec back to you in time to do this. If we do, though, you

MUST as described above turn on Microsoft Word change tracking bars BEFORE beginning updating the old requirements spec.

In the Operating Procedures part of the new user manual, make sure you illustrate the use of the system using sample data. Using examples is extremely illuminating to a user, and is also necessary to get good feedback from a initial customer/user review of the manual. This will allow revisions to the external design to take place before the architectural design and coding start. i.e. We are prototyping on paper. Note that example use cases with sample data are not wasted effort as you will need to make up test cases containing such data for one of the later assignments anyway!

It is necessary to document exactly what will happen to the user if bad data is entered. These are called operational exceptions. To reduce your workload in Cmpt 275, I will not require you to document/handle input errors in data format and range, *except those automatically returned by the I/O routines* (e.g. end of file, entering "abc" to an integer read). Note that in a real application there is often a lot of input data checking. I will however, require handling errors generated by the I/O functions and I also expect to see in the user manual how the system will handle cases of:
a) trying to create an item when one with the same key already exists.
b) trying to create an item and there is not enough disk space.
c) deleting an item and it doesn't currently exist.
d) referring to an item which should exist but doesn't (e.g. for a ferry reservation system adding a reservation to a sailing when the sailing doesn't exist). This ensures what is called 'referential integrity'.

You must provide an exact illustration of the exceptional behavior of the system in these cases (e.g. screen illustration of the exact error messages). It does not have to be screen dump, just an illustration or even just the wording of the error message, and importantly, an indication of what the user can do next. The later is necessary to indicate to the coders whether they should allow the user to enter data again, and if entering again unsuccessfully, how to abort the command! Nothing drives a user more crazy than not being able to exit out situation.
So the manual (or screen wording) should suggest how to correct and/or get around the problem. Note that if the same exception occura from several different user operations, you may want to detail the exceptional behavior in one place (e.g. in an error reference section or appendix) rather than duplicate this info at every location where that error can occur. In this case, each location will refer the user to the appropriate part of the error reference section. Another good thing to put in the reference section is a menu tree (or it could go in the User Interface description section as an intro to the UI).

You must plan and tell users how to back out of an operation if they decide not to do it. If you have a common way of doing this for all operations, you need explain it just once in the UI section.

Remember who the reader will be: a typically user. For a ferry system or flight school, that might be a dispatcher who may never have operated a computer before. For a bug and feature request tracking system, it may be a customer service representative who has little computer experience. For an airfreight system, it might be the loading clerk. You must explain how the user interface works (e.g. does every entry need to be followed by a <CR>, or just some of them? Which ones?) Will the backspace key work during some or all data entries? Students have in the past been unclear whether the user interface was a scrolling system which prompts you for functional data line by line, or a character-addressable screen where you move from field to field using <TAB> (typically they meant the first but illustrated screens for the latter). If it is not possible to differentiate visually between these two cases in your user manual, at least comment on what your system uses or else you will loose marks! You may assume a 'slightly' more sophisticated person is installing the software. In addition, assume the user has <u>not</u> read the assignments or Requirements Spec; this is very important and relatively hard to actually do well because it is hard to imagine being a person who has not read these when you have read them! And, as in all Cmpt 275 projects, spelling, grammar, style for the correct audience, and clarity count, particularly for Assignment #2!

Please remember when the program is first started up after installation, the data files may not exist. Thus when your program tries to open them, it may encounter a file open failure. Therefore, you must either:
1) detect this in your code (it's not hard), and do file creates as necessary, or
2) instruct the user in the user manual to run the Initialize menu item which (dangerously) deletes any existing files and creates fresh new empty ones, or
3) tell him/her that empty or example data files are shipped with the system.
You may choose either of these options for Cmpt 275.

What you hand in will be a (signed) release title page, the release history and release table pages, followed by an updated requirements spec, then the user manual. The user manual itself must have a (signed) title page, and of course a revision history page and table of contents. There must be **labelled-tab dividers before each CSCI item or you will loose marks**.

Hints for all semesters:

- Make a copy of the full ORD, remove the aircraft model class and make other changes as necessary to reflect the lack of a model, then put this reduced ORD in the prototype section of the Requirements Spec. Leave a full ORD in the main part of the Requirements Spec.
- You needn't bother for Cmpt 275 putting boxes around all screen examples to simulate looking at a DOS-like console screen or terminal. Instead, just use different fonts (fixed space fonts such as `Courier` are best as dumb terminals use 80 fixed-space characters across the screen). You may even want to use two different fonts for the screens, one for prompts and a bolded one to indicate user inputs.
- One way to reduce later implementation effort is to realize that item inquiries can be done via a delete operation which would display all the item's existing data, and then ask if you want to actually proceed to delete this one. Though this is dangerous for normal users, for CMPT 275 it will save you having to implement a separate Inquire operation for objects you have been told to provide a delete operation for.
- Explain any possible ramifications of telling the user to modify things via a delete then re-add, and the problems of deleting objects that are currently involved in relationships.
- Many of you do not know how to print directly from a program in your favourite operating system. So, instead, when doing a printed report, tell the user in the manual that the report will be written to a file with a particular name and they should then use their standard OS user commands to get that file printed to the printer.

You will be marked out of 50 as follows:
3 marks- release pages and updated older CSCIs.
3 marks- user manual title, history, and TOC pages.
2 marks- overview of application.
2 marks- seemingly realistic and definitive installation guide.
12 marks- operating procedures including general explanation of the UI, a sensible menu tree design, the good design of prompts, well explained error handling, and examples of screen and data layouts.
2 marks- back-up procedures.
6 marks- completeness - all screens and formats, required procedures, recommendations and hints to users.
5 marks- reference section and glossary.
5 marks- understandable by novice user.
5 marks- spelling and grammar.
5 marks- organization and neatness (don't forget the labelled tab dividers!).

50 total